Truck Platooning: week 5

Changyao Zhou, Jiaxin Pan

January 24, 2022

1 Introduction of the project

The goal of this project is to chase a chosen target vehicle by predicting the steering angle and throttle input of the ego-vehicle, given the velocity of both vehicles and images taken from cameras.

The control values (steering angle and throttle) computed by Model Predictive Control (MPC) are used as ground-truth. MPC is an advanced method of process control that is used to control a process while satisfying a set of constraints[WIK]. It takes a prediction horizon instead of a single time step into account, and aims to get an optimal control result minimizing the objective function within the prediction horizon. For learning the MPC controller, we take [MPC] as reference.

The project is designed to be realized in the following steps:

1). Control the ego-vehicle with Model Predictive Control (MPC) in the Carla simulator

2). Given x, y-location, yaw angle and velocity of both vehicles, train a fully connected neural network to control ego-vehicle by using the MPC output as ground-truth.

2). Given object velocity, ego-velocity and RGB image, train a neural network to control the ego-vehicle by using the MPC output as ground-truth

3). Further improvement by changing the input of the neural network

2 Overview of this week

Last week we have trained our first fully-connected neural network with the input information of the x-/y-positions, yaw-angles and velocities of the both target- and ego-vehicles. The result was quite stable. We further tried our first FCNN with two following vehicles, and it worked well. The result can be checked here: https://syncandshare.lrz.de/getlink/fiHWciH4LfHwaa47Q2vGUpgw/model_test_2fol.mp4.

This week we first tried to train a second FCNN with the differences of the x-/y-positions, yaw-angles between two vehicles and two velocities as input. The training data are calculated based on the data we collected last week.

After that, we tried to train a CNN(convolutional neural network) with images collected in the same 'online' mode in the Carla simulator.

All the videos are stored under https://syncandshare.lrz.de/getlink/fiSekEyJ5uaMK8ArnLN4uanu/week5.

3 Training of the second FCN Model

To train the network, first we computed the inputs based on the data we collected last week, which means we used the differences between the states of two vehicles as inputs. The labels are the same.

The training of this FCNN seemed to be harder than the first FCNN, probably because there are less input information. We first tried with the structure that produced the best result in last week, however, after 6000 epochs, **the training loss seems to be very low while the validation loss still quite high and unchanged for the next few epochs, which means the model has already overfit the training dataset.** Apart from this network structure, we have tried other structures with less/more layers, or with other activation functions such as LeakyRelu, or with dropout. The

network with more complicated structure always got stuck with the overfitting problem and the easier-structured networks had problems of underfitting. So we could not find a structure that produces the result as perfect as in last week.

In last week we got a validation loss of 9e-2, with the FCNN in this week, the final validation loss was about 3–3.5e-1, which is 3 times higher than last week.

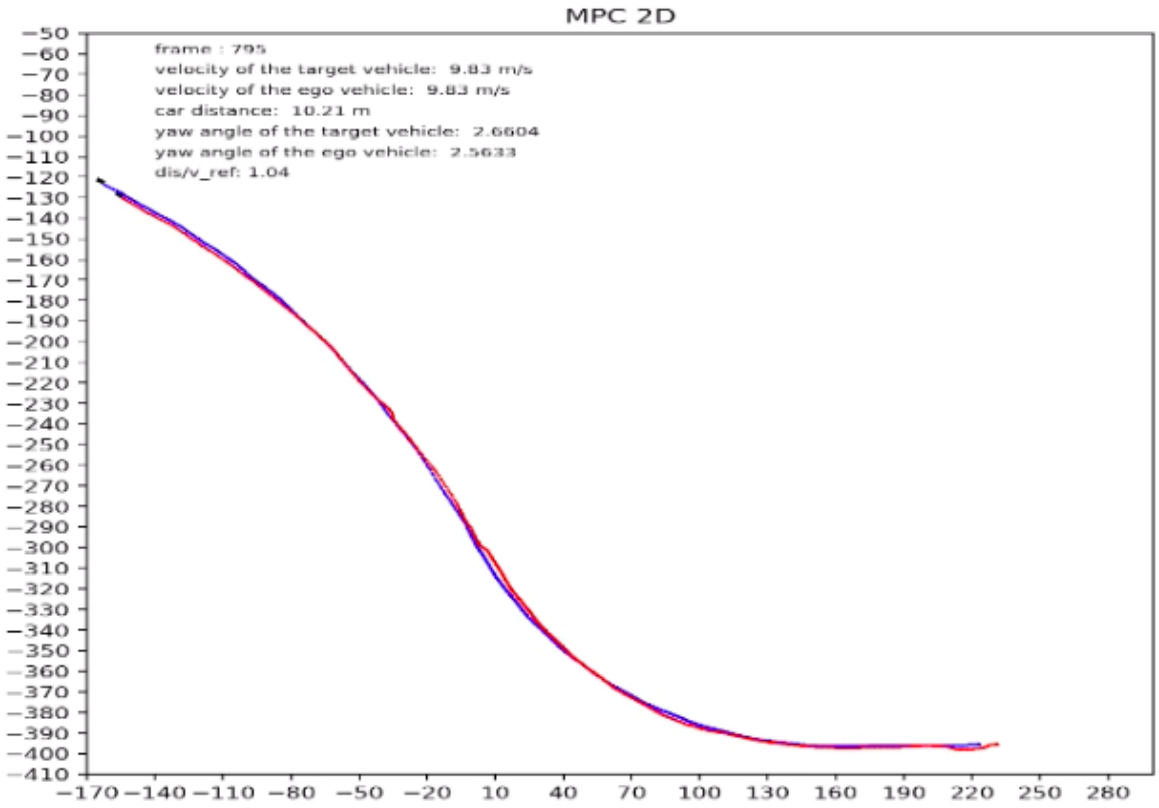

Figure 1: The inference test in Carla simulator.

After training, we tested the inference again in Carla simulator. But the inputs this time are the Δx, Δy, Δyaw and Δv

The video for car following with model inference can be found here: https://syncandshare.lrz.de/getlink/fiTABjKSfpG6AbNdTam2FtEX/carla_model_delta.mp4

As can be seen in the Figure 1 and in the video, the ego vehicle can follow the target vehicle most of the time, but it's not that stable as in the first model. Especially the steering angles are not that accurate and leads to relatively large yaw errors.

4 Training of the CNN Model

4.1 Date Preparation

After the training with FCNN, we moved to CNN models. The images are collected with an RGB camera attached to the ego-vehicle and placed at the front. We collected images both on-trajectory and off-trajectory with the 'online' mode, exactly the same as we collected the information of the states.

The Figure 2 is one of the images collected from the RGB camera, as shown in the figure, the target vehicle (in red) can be clearly seen, also part of the ego vehicle (in black) is visible. So the network can tell the difference in yaw angles between the two vehicles from the images. In total we have 30693 samples, 24556 are used for training and 6137 are used for validation/test.

Figure 2: An image from the camera.

4.2 Overfitting with 4 samples

Since the whole dataset is very large and takes long to train it, we first tested different structures with a small set of 4 random samples. The network is made up with 2 parts, the first part is a CNN, which extracts features from the images. The velocities of the two vehicles are then concatenated to the extracted features and are passed as input of the second part. The second part is a FCNN, processing the extracted information and making final predictions.

For the first CNN, we tried different pretrained net with the overfitting set, to test which produces the smallest loss.

Table 1: Training result of the 4-sample dataset

	Alexnet	vgg16	vgg16_bn	densenet169	mobile net_v3_small
parameters	14,226,498	28,926,530	28,926,530	58,546,818	20,163,618
training loss	1.311578e-10	7.666825e-11	8.905064e-12	2.422443e-10	4.306764e-04
validation loss	4.256948e-10	5.501847e-11	1.341233e-11	1.089751e-10	4.303502e-04
200 epochs time	30”	1’45”	1’56”	2’17”	35”

The result is shown in Table 1. All structures except the pretrained mobilenet produced quite small loss. Since the pretrained Alexnet has the least parameters and leads to a fast training, we used the pretrained Alexnet to extract features from the images.

4.3 Training Result

With the pretrained network, less epochs are needed for training. We trained with the whole dataset for about 50 epochs and the loss is already stable, which is much less than the 6000 epochs for training the FCNN.

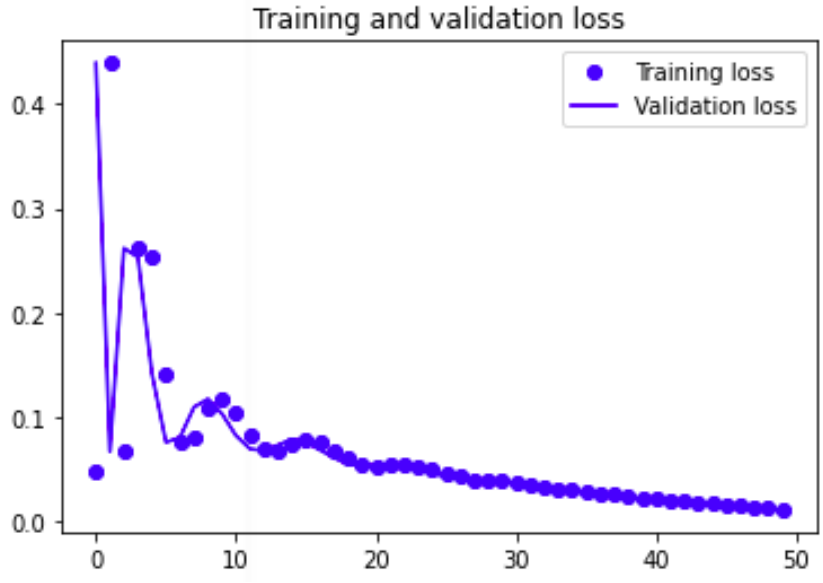

Figure 3: The loss curve during 50-epoch training

The loss curve is shown in Figure 3. The final loss is about 3.7e-3. To test the result, we put the model into the Carla simulator.

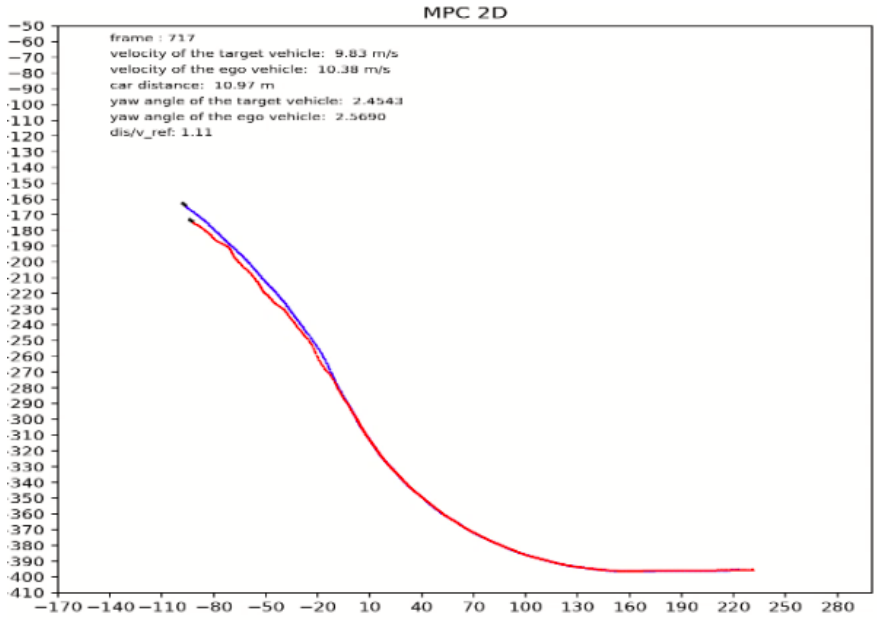

Figure 4: The inference with Carla simulator.

We made two tests,in one of them the target vehicle ran exactly as in the training set and in another a little different. The result of the second one is shown in Figure 4. The video can be checked here: https://syncandshare.lrz.de/getlink/fiGMSrCvkaMrZarv84GBAmU6/inference_CNN_on.mp4 and here:https://syncandshare.lrz.de/getlink/fiP2vvpS2Hbk3AtDsafMQuHq/inference_CNN_off.mp4.

As can be seen from the figure, the predicted control values were quite reasonable in the first 590 frames than the ego vehicle became unstable. That is probably because that in the training set the target vehicle was on the left lane, but in the test the target vehicle was on the right lane and there maybe sometimes that the target vehicle is not visible for the ego vehicle. That makes the predictions become relatively harder.

To test the sability of the model, we added some random impulses on the ego vehicle. The result is shown in Figure 5. The video can be checked here: https://syncandshare.lrz.de/getlink/fiDJ3npAdcgZd8ZeGR7m71SR/carla_model_CNN_impulse.mp4

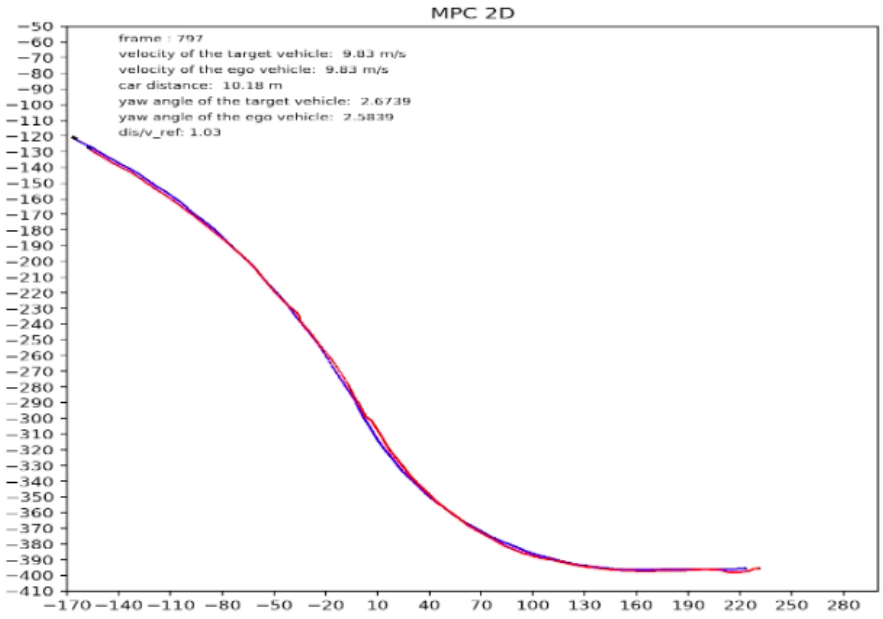

Figure 5: The inference with random impulses.

As can be seen from the image. With the random impulses, in most cases, the model can predict control values that let the ego vehicle back to right position, but it was not as stable as in the last week.

5 Conclusion and future work

In this week, we first tried to train a FCNN with the Δstates of the two vehicles and the it seems that with these input information, the ego vehicle can follow the target vehicle, but not as perfectly as putting the raw states as input.

After that, we collected the images from an RGB camera attached to the ego vehicle and trained the first CNN model with pretrained Alexnet. The result seems to be quite good, but not that stable in comparison to the result in last week.

In the next step, we will further train the CNN to produce more reasonable results.

References

[MPC] Model Predictive Control. https://github.com/WuStangDan/mpc-course-assignments.

[WIK] Model Predictive Control. https://en.wikipedia.org/wiki/Model_predictive_control.